

# Java Programming 2 – Lab Sheet 3

This Lab Sheet contains material based on Lectures 5—6.

**The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 17 October 2019.**

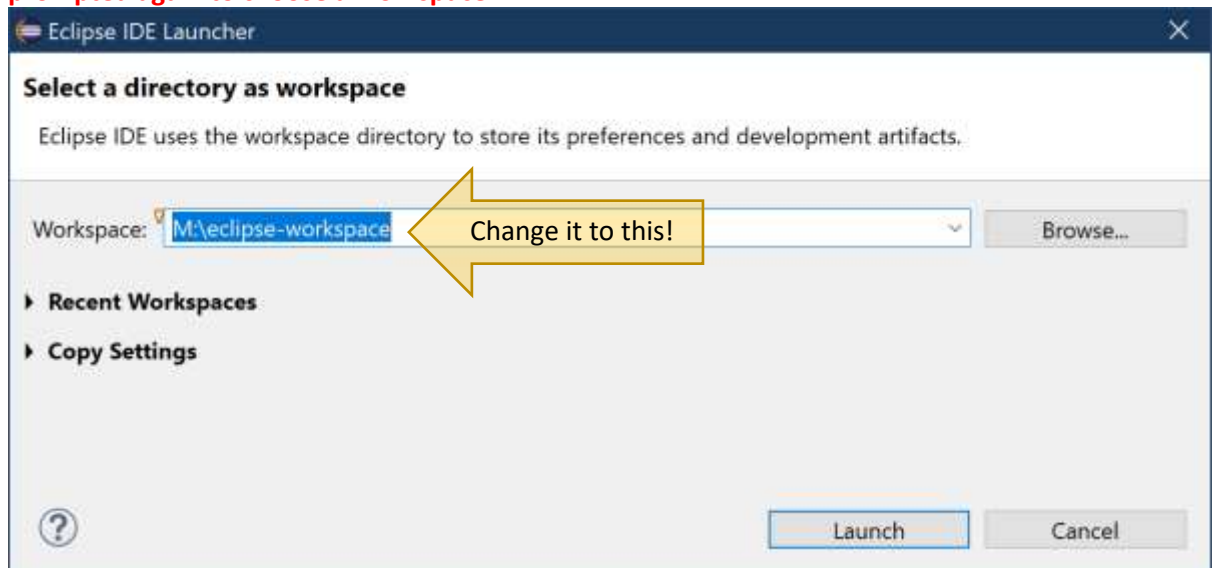
## Aims and objectives

- Using the Eclipse IDE to develop more complex Java code
- Writing Java classes including fields and methods

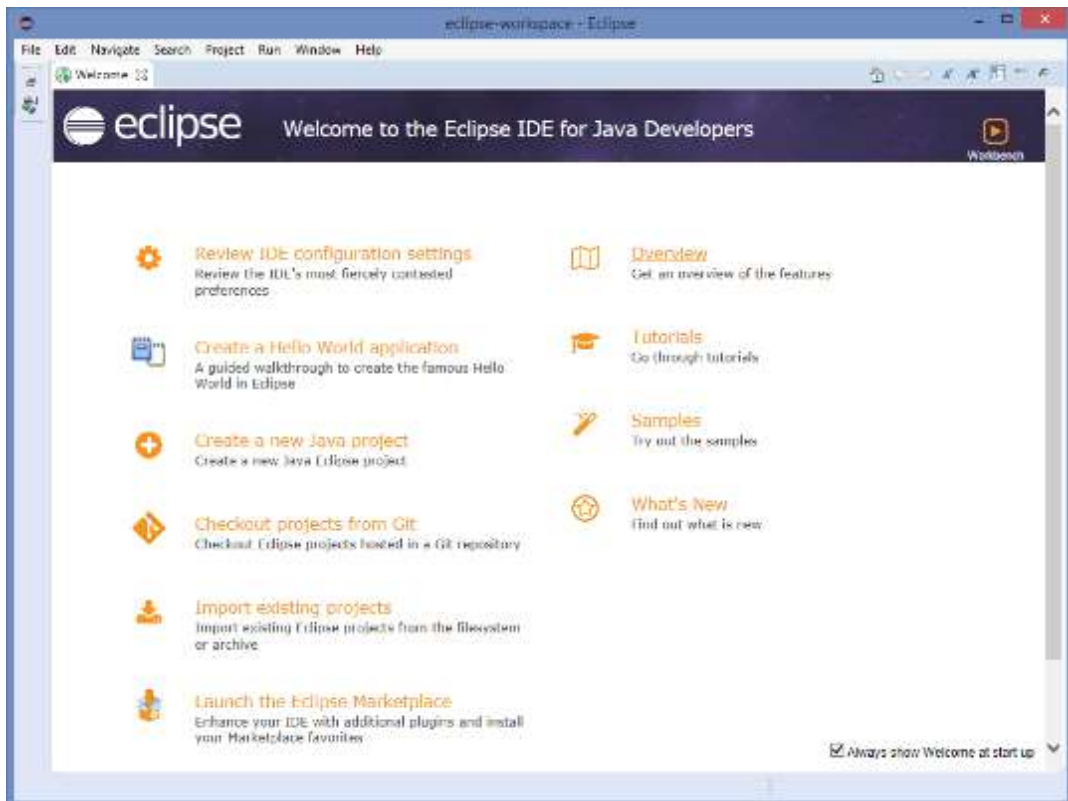
## Using Eclipse

Eclipse is an open-source Integrated Development Environment (IDE) that will be used for coding, compiling, running, and debugging Java programs. Eclipse provides many more facilities than this, some of which you will use in later courses. It is a many-featured and versatile piece of software, and can appear a little daunting for beginners. We will be using only a small subset of its capabilities, and a key objective of this lab session is to begin the process of familiarisation with these.

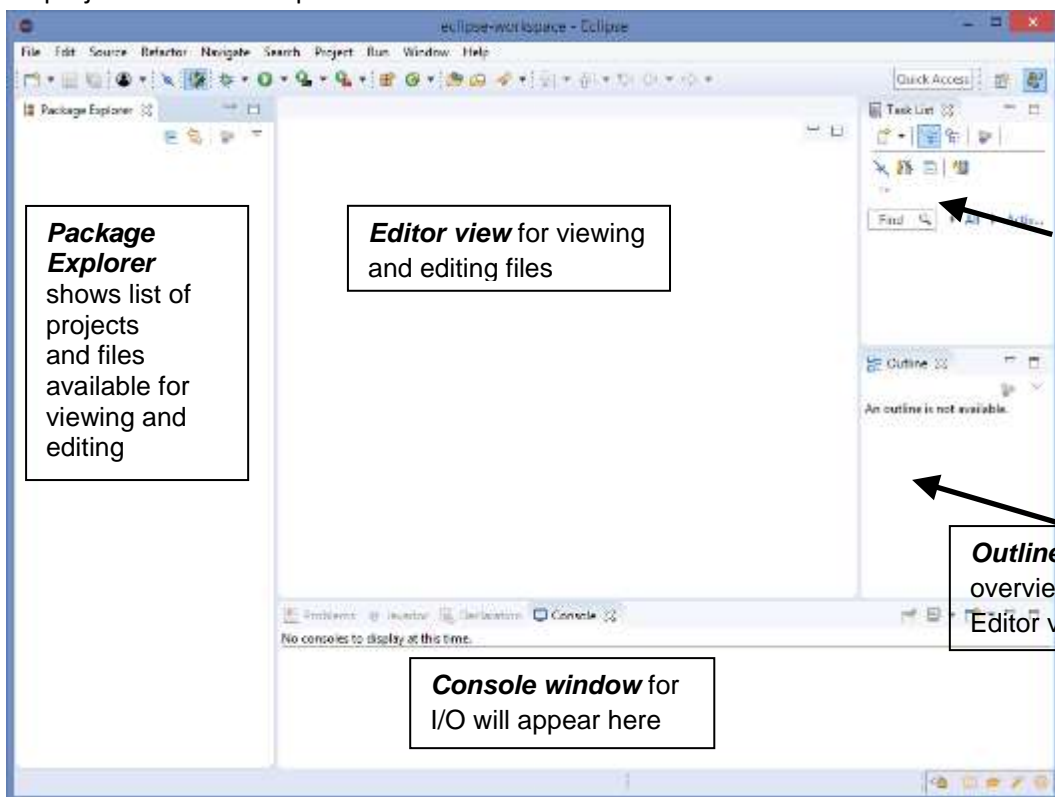
- Launch Eclipse by clicking the Windows menu and choosing “Eclipse Java”. After it launches, you will first be asked to choose a **workspace folder** – this is the folder that will contain all of the files you are working on. **When you run Eclipse, you MUST modify the “Workspace” field to point to a directory under your own file space, for example “M:\eclipse-workspace”, or else your work might not be available when you log into a different machine. You should also tick the “Use this as default and do not ask again” to avoid being prompted again to choose a workspace.**




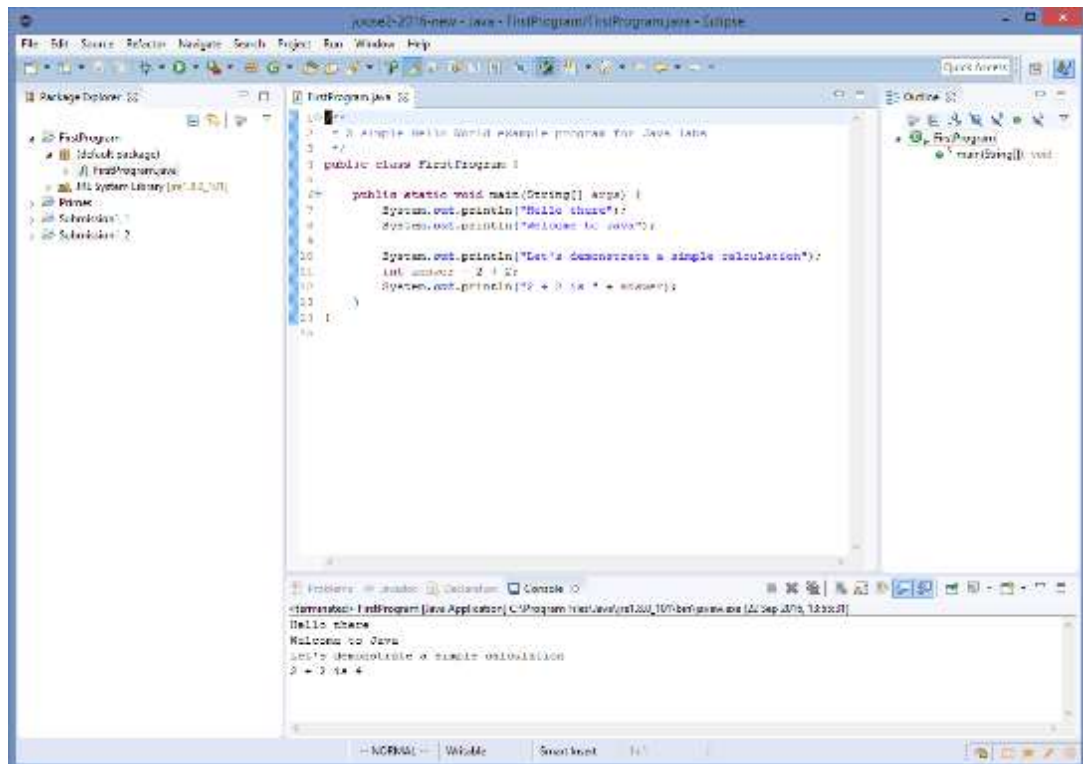
- Eclipse takes a few seconds to start up. You will see Eclipse's 'welcome' screen appear - see below. You can un-tick the “Always show Welcome at start up” box to ensure that you go straight to the workspace in future.



- Clicking on the Workbench arrow (arrow in box on top right hand side) will take you to the main IDE. The screenshot below shows the appearance of the default Java perspective when no projects or files are open.



- Each Java program that you work on will form an Eclipse project, and this project has to be created. Typically, some or all of the files that are required for a project will be downloaded, at least in skeleton form, when you download the lab zipfile from moodle. Each project will have its own folder, the name of the folder necessarily being the same as the name of the project. You will need to import the project(s) into your workspace so that you can run and modify them.
- In Eclipse, select **File** → **Import** ... to launch the import wizard – this will be used to import the starter projects into your workspace. Then select **General** → **Existing Projects into Workspace** and click **Next**.
- Choose **Select archive file** and then browse to the location where you downloaded `Laboratory3.zip`. Select `Laboratory3.zip` and click **Open**.
- You should now see a set of two projects in the Projects window: **FirstProgram** and **Lab3**. Ensure that the checkboxes beside both of these are selected (e.g., by pressing **Select All** if they are not), and then press **Finish**.
- In the **Package Explorer** view (on the left of the Eclipse window), you should now see icons representing the three projects. Clicking on the expander icon  beside **FirstProgram** reveals the contents of the project – namely components of the Java Run-Time Environment (JRE) together with the *default package*, which contains the sole source code file for this program, i.e. `FirstProgram.java`. Expanding the default package icon will reveal this file. Then double click the file name to open the file in the central **Editor view**. You will see the source code for the 'Hello World' program, and in the **Outline view** you will see a summary (brief in this case) of the methods in this class.
- Now click on the **Editor view** to activate it. Whenever Eclipse detects an error in the java file displayed in the editor view, this is indicated by red underlines in the text, and red crosses to the left (and possibly also the right) of the view. Hovering the cursor over a marker throws up an error message – see below. This feature of Eclipse takes a bit of getting used to; it can be annoying as the compiler often does not give you time to finish typing a line before throwing up some spurious error indicator, but in general this can be a very useful feature once you get used to it.
- The program here is complete and contains no errors, hence Eclipse does not complain. But nonetheless, it has done its work behind the scenes, and you can run the program. To do this, first make sure that the **Editor view** is active (as indicated by a blue border). Then select the Menu option **Run** → **Run as** → **Java application**, or click the green 'Play' button in the toolbar. You should see the output from the program in the **Console view**, at the foot of the Eclipse window.



- Now introduce an error into the Java code. For example, delete one of the letters from the word `static`. You will see a red cross appear on the extreme left of the Editor view, and moving the cursor over this marker reveals a helpful error message. (Error messages are not always as immediately helpful as this.) Experiment with some further errors – for example, try deleting punctuation marks like semicolons or curly brackets to see what errors are indicated.
- You may have noticed that when the cursor hovers over an identifier (in the **Editor** view) information about that identifier is displayed. You will discover a variety of useful features of the Eclipse editor in due course.
- You are now ready to work on the programming exercises below that form the submission material for this lab.

## Submission material

You are to design and implement a class **Monster** representing a (simplified) monster in a monster-battling game. A monster includes a **type**, a number of **hit points**, a list of **attacks**, and a list of **attack points** (i.e., each attack has a corresponding number of attack points).

In the Eclipse project Lab3, I have provided a file **Monster.java** which contains only the declaration of the **Monster** class and an empty body. You should fill in the class body by implementing all of the fields and methods described below.

## Fields and constructor

The relevant properties should be included in the **Monster** class by defining the following private fields:

- `type` (a `String`)
- `hitPoints` (an `int`)
- `attacks` (a `String[]`)
- `attackPoints` (an `int[]`)

There should be a single public constructor method that takes four parameters to initialise the four fields. That is, the constructor signature should be:

```
public Monster(String type, int hitPoints,  
               String[] attacks, int[] attackPoints)
```

For this lab, we will only test your constructor with valid input for the above parameters, so **you do not need to include any input validation** (e.g., we will only test with positive hit points, and will only use arrays of attacks and attack points that have the same length).

## Methods

Define getter methods for all of the **Monster** fields:

- `public String getType()`
- `public int getHitPoints()`
- `public String[] getAttacks()`
- `public int[] getAttackPoints()`

Also override the inherited `toString()` method to return a `String` including the values of the object fields. The format is up to you; however, you must be sure to include at least the type, hit points, and details of the attacks.

You should also define the following two methods which are used when monsters battle each other:

- `public void removeHitPoints (int pointsToRemove)`
  - Removes the indicated number of hit points from the current monster. If the hit point value becomes negative, then it should be set to zero.

- `public boolean attack (String attack, Monster otherMonster)`
  - This method is called when the current monster attacks the other monster with the given attack. It should proceed as follows:
    - If **otherMonster** is actually the same as this monster (use “==” to check), return false
    - If either the current monster or the other monster is knocked out (i.e., hit points == 0), return false
    - If the monster does not have an attack with the given name, return false
    - Otherwise:
      - Find the number of attack points corresponding to the given attack
      - Remove that number of hit points from the other monster by calling `removeHitPoints()` on it

## Examples

To test your code, you can add a `main()` method to your **Monster** class to allow it to be run in Eclipse; you can then create, modify, and print out **Monster** objects. There is an example of this process in the **BankAccount** class. But if you do this, **please remember to remove the `main()` method before submitting your code** as it is not part of the specification for the **Monster** class.

For example, you might want to try the following code – you can also try other things. Be sure that you have tested all of the above behaviour and that it behaves as specified.

```
Monster moltres = new Monster("Fire", 114,
    new String[] { "Fire Spin", "Overheat" },
    new int[] { 14, 160 });
Monster articuno = new Monster("Ice", 112,
    new String[] { "Frost Breath", "Ice Beam" },
    new int[] { 10, 90 });
Monster zapdos = new Monster("Electric", 119,
    new String[] { "Charge Beam", "Thunder" },
    new int[] { 8, 100 });

System.out.println(moltres);
System.out.println(articuno);
System.out.println(zapdos);

System.out.println(moltres.attack("Fire Spin", articuno)); // true
System.out.println(moltres.getHitPoints()); // 114
System.out.println(articuno.getHitPoints()); // 98

System.out.println(moltres.attack("Overheat", zapdos)); // true
System.out.println(zapdos.getHitPoints()); // 0

System.out.println(moltres.attack("Bad Attack", articuno)); // false
System.out.println(articuno.getHitPoints()); // 98

System.out.println(articuno.attack("Ice Beam", articuno)); // false
System.out.println(articuno.getHitPoints()); // 98

System.out.println(zapdos.attack("Thunder", moltres)); // false
```

## Hints and tips

1. The sample **BankAccount** class (included in the **Lab3** Eclipse project) gives a sample of most of the things that you will need to implement in your **Monster** class.
2. Make sure that none of your methods produce any output through `System.out.println` or similar mechanisms unless specified. You are implementing a class that will (in theory) be used as part of a larger system, so output is not appropriate unless specifically requested.
3. Once you have defined the fields of your class, Eclipse is able to automatically generate many of the methods that you will need for this assignment based on those fields. For example, try right clicking on the source file and choosing “Source – Generate getters and setters ...”, “Source – Generate constructor using fields”, or “Source – generate toString()”. Experiment and see what it generates!
  - If you do this, you **\*\*MUST\*\*** read through the automatically generated code to be sure that it behaves as required – any automatically generated code should be thought of as a starting point only.
4. To convert an array to a nice string representation, you can use **Arrays.toString()** as follows (assuming that **arr** is an array variable):
  - `String arrayStr = Arrays.toString (arr);`
5. Eclipse is able to automatically clean up code formatting and indentation – if you press **Ctrl-Shift-F** or go to **Source – Format** then your current source file will be reformatted. I strongly suggest doing this before submitting to ensure that you do not lose any marks for code style.

## How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not. Before submission, make sure that your code is properly formatted (e.g., by using **Ctrl-Shift-F** to clean up the formatting), and also double check that your use of variable names, comments, etc is appropriate. **Do not forget to remove any “Put your code here” or “TODO” comments!**

When you are ready to submit, go to the JP2 moodle site. Click on **Laboratory 3 Submission**. Click ‘Add Submission’. Open Windows Explorer and browse to the folder that contains your Java source code – probably **M:\eclipse-workspace\Lab3\src\** -- and drag only the *one* Java file **Monster.java** into the drag-and-drop area on the moodle submission page. **Your markers only want to read your java file, not your class file.** Then click the blue save changes button. Check the.java file is uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via moodle.

## Outline Mark Scheme

Your tutor will mark your work and return you a score in the range “Excellent” (\*\*\*\*\*) to “Very poor” (\*). We will automatically execute your submitted code and check its output; we will also look at the code before choosing a final mark.

Example scores might be:

**5\*:** you complete the code correctly with no bugs and correct style

**4\*:** you compute the class mostly correctly with minor bugs OR your code is correct but the style is inappropriate

**3\*:** more major bugs and/or more major style issues

**2\*:** some attempt made

**1\*:** minimal effort

For this assignment (and all future ones), we will also be considering code style – comments, formatting, variable names, etc – in addition to correctness. You will be deducted one star if your code style is not appropriate. Please check with the tutors and demonstrators during your lab session if you are uncertain about style, or look at the BankAccount class for an example.